A domain function for MAPLE

by

Mark Hickman
Department of Mathematics, University of Canterbury,
Christchurch, New Zealand

No. 78 March, 1993.

Abstract. A collection of MAPLE V Release 2 routines are presented that implement a "domain" type function (similar to the depend function of REDUCE [1]) for MAPLE. This avoids the tedious requirement of always specifying the functional dependencies when using MAPLE. Further, an extension of dsolve is presented that exploits this domain function. The domain function is also exploited in the author's package to determine symmetries of differential equations [2].

A domain function for MAPLE

Mark Hickman
Department of Mathematics
University of Canterbury
Christchurch, New Zealand
email: msh@math.canterbury.ac.nz

Abstract

A collection of MAPLE V Release 2 routines are presented that implement a "domain" type function (similar to the depend function of REDUCE [1]) for MAPLE. This avoids the tedious requirement of always specifying the functional dependencies when using MAPLE. Further, an extension of dsolve is presented that exploits this domain function. The domain function is also exploited in the author's package to determine symmetries of differential equations [2].

1 Introduction

MAPLE requires that the functional dependencies of the argument be given explicitly in each call to, for example, `diff`. In practice this is tedious and one would normally want to suppress the arguments (when working with "pencil and paper" one normally doesn't include the functional dependencies). This can be implemented via the use of the `alias` command.

During the development of the author's symmetry package [2], it was noticed the use of the MAPLE function data type (that is `A(x)`, for example) results in single level evaluation rather than the usual full evaluation. The reason for this is that MAPLE sees the function argument as a parameter and thus does not regard `A(x)` as a global variable. This has very undesirable consequences as illustrated in the following MAPLE session.

> a:=diff(A(x,y,z),x);

$$a := \frac{\partial}{\partial x} \mathrm{A}(x, y, z)$$

> A(x,y,z):=B(x,y);

$$\mathrm{A}(x, y, z) := \mathrm{B}(x, y)$$

> B(x,y):=C(x);

$$\mathrm{B}(x, y) := \mathrm{C}(x)$$

> C(x):=x^2;

$$\mathrm{C}(x) := x^2$$

> a;

$$\frac{\partial}{\partial x} \mathrm{B}(x, y)$$

> eval(a);

$$\frac{\partial}{\partial x} \mathrm{C}(x)$$

> eval(eval(a));

$$2\,x$$

The fact that even the explicit use of `eval` will not force full evaluation would create problems in any process that involves a long chain of substitutions (for example, the computation of symmetries). Of course the non-inert differentiation, `diff`, requires arguments in order to determine functional dependencies and so some type of data structure is needed that will allow the functional dependencies to be encoded. The problem with evaluation, fortunately, does not occur with *indexed* quantities of the type `A[x]`. Furthermore, `diff` does recognize the indicies as functional dependencies. The procedure domain uses the indexed data type to represent functions. The code may easily be changed to use the function data type if required (see fdomain in Appendix A).

2 Implementation

The procedure domain gives a convenient way to attach functional dependencies to labels. The call domain(vlist, f, g, ...) will declare f, g, ... to depend on the variables contained in the list vlist. The dependencies are stored in a global variable _FUNCNAMES so that the dependencies can be restored via explicit. If a label has already been given a functional dependency then domain will warn the user of this fact and ask if the label is to be redefined. When redefining the functional dependencies, the new variables should be a subset of the original set of variables. This is to avoid the following situation. Suppose f is declared to depend only on x and a:=diff(f,t) then, even if f is redefined to depend on t, a would still evaluate to 0. In this case domain will change the dependencies but will also issue a warning. Note that if a label has already been assigned a value then it can not be given declared though domain. The procedure var will give the dependencies of its argument. If it is call with no arguments then it gives all the current function declarations.

```
_FUNCNAMES:=NULL:

domain:=proc(vlist:list) local i,pos,tmp,tmp1,vars;
options `Copyright 1993 by Mark Hickman`;
vars:=op(vlist);
for i from 2 to nargs do
tmp:=args[i];
if tmp = explicit(tmp) then
   _FUNCNAMES:=_FUNCNAMES,tmp=tmp[vars];
   alias(tmp=tmp[vars])
else
   tmp1:=op(0,tmp);
   convert(tmp1,string);
   print(cat(",` has already been declared`));
   print(cat(`Do you wish to redefine `,
             ",`? y or n`));
   if readline() = 'y' then
```

```
      if not {vars} union {op(tmp)} = {op(tmp)} then
         print('WARNING: The new variables');
         print(vars);
         print('are not a subset of the old variables');
         print(op(tmp))
      fi;
      member(tmp1=tmp,[_FUNCNAMES],'pos');
      _FUNCNAMES:=op(subsop(pos=NULL,[_FUNCNAMES]));
      alias(tmp1=tmp1);
      assign(tmp=tmp1[op(vlist)]);
      _FUNCNAMES:=_FUNCNAMES,tmp1=tmp1[op(vlist)];
      alias(tmp1=tmp1[op(vlist)])
   fi
fi
od end:


var:=proc(fname)
options 'Copyright 1993 by Mark Hickman';
if nargs = 0 then explicit([_FUNCNAMES]) else op(fname) fi end:

explicit:=proc(x)
options 'Copyright 1993 by Mark Hickman';
subs(_FUNCNAMES,x)  end:
```

In the procedure `domain`, the local variable `tmp` represents the aliased quantity whereas `tmp1` represents the label *only*. In the output both `tmp` and `tmp1` would appear the same but they are not the same to MAPLE (this can be checked by the `addressof` command or, in this case, by the fact that they are different data types). This distinction is particularly important when an already current alias needs to be changed. The information of the current aliases is stored in `_FUNCNAMES` in the form of `label = alias`. In the output, `_FUNCNAMES` appears to be an expression sequence whose elements are of the form `f = f`.

The procedure `desolve` "extends" `dsolve` to handle differential equations involved indexed quantities rather than functions. `desolve(eqn,f,x, ...)` will attempt to solve the equation `eqn` for `f` in the variable `x`. Unless it is called with other arguments, the arbitrary "constants" will be labelled `_f1, _f2` and so on. If further arguments are given then these will be used for the constants. The constants are declared to be functions of all the variables of `f` except `x`. If the result is an explicit expression for `f` then `desolve` will assign `f` this value otherwise the implicit form will be echoed on the terminal.

```
desolve:=proc(eqn,u,x) local tmp,tmp1,tmpvar,i,constname;
options 'Copyright 1993 by Mark Hickman';
tmp:=dsolve(subs(u=u(x),eqn),u(x));
if tmp=NULL
    then RETURN('Can not solve this differential equation')
else
    tmp1:=NULL;
    i:=1;
    tmpvar:=sort([op({var(u)} minus {x})],varorder);
    while not tmp1 = tmp do
       tmp1:=tmp;
       if nargs > (i + 2) then constname:=args[i+3]
       else constname:=cat('_',op(0,u),i) fi;
       if tmpvar=[] then
          tmp:=subs(_C.i=constname,tmp1)
       else
          domain(tmpvar,constname);
          tmp:=subs(_C.i=constname[op(tmpvar)],tmp1)
       fi;
       i:=i+1
    od
fi;
subs(u(x)=u,tmp);
if op(1,")=u then
   assign(");
   eval(u)
else eval(")
fi
end:
```

The procedure `desolve` can be used as a model to convert other MAPLE calls that assume a function data type. If a function is declared using `fdomain` then this should not be needed. The following is the record of a MAPLE V Release 2 session illustrating the use of these routines. A far deeper application appears in [2].

> domain([x,t],f);

$$I, f$$

> diff(f,x);

$$\frac{\partial}{\partial x} f$$

> diff(f,z);

$$0$$

> int(f,x);

$$\int f \, dx$$

> explicit(");

$$\int f_{[x,\,t]} \, dx$$

> int(f,z);

$$f\, z$$

> eqn:=diff(f,x$2)+f;

$$eqn := \left(\frac{\partial^2}{\partial x^2} f \right) + f$$

> desolve(eqn,f,x,A,B);

$$A \sin(x) + B \cos(x)$$

> explicit(");

$$A_{[t]} \sin(x) + B_{[t]} \cos(x)$$

> f;

$$A \sin(x) + B \cos(x)$$

> diff(f,t);

$$\left(\frac{\partial}{\partial t}A\right)\sin(x)+\left(\frac{\partial}{\partial t}B\right)\cos(x)$$

> **fdomain([x,t],g):**
> **diff(g^2,x);**

$$2\,g\left(\frac{\partial}{\partial x}g\right)$$

> **explicit(");**

$$2\,\mathrm{g}(x,t)\left(\frac{\partial}{\partial x}\mathrm{g}(x,t)\right)$$

> **taylor(g^2,x=0,3);**

$$\mathrm{g}(0,t)^2+2\,\mathrm{g}(0,t)\,D_{[1]}(g)(0,t)\,x+$$
$$\left(\mathrm{g}(0,t)\,D_{[1,1]}(g)(0,t)+D_{[1]}(g)(0,t)^2\right)x^2+\mathrm{O}(x^3)$$

> **domain([x,t],h):**
> **domain([y],h):**

h has already been declared

Do you wish to redefine h? y or n

y

WARNING: The new variables

y

are not a subset of the old variables

$x,\,t$

> **var();**

$$\left[A=A_{[t]},\,B=B_{[t]},\,\_f3=\_f3_{[t]},\,g=\mathrm{g}(x,t),\,h=h_{[y]}\right]$$

A Using the function data type

The code of domain is rewritten so that it uses the function data type rather than the indexed data type. The procedure has been renamed fdomain. Note that explicit and var remain unchanged.

```
_FUNCNAMES:=NULL;

fdomain:=proc(vlist:list) local i,pos,tmp,tmp1,vars;
options `Copyright 1993 by Mark Hickman`;
vars:=op(vlist);
for i from 2 to nargs do
tmp:=args[i];
if tmp = explicit(tmp) then
   _FUNCNAMES:=_FUNCNAMES,tmp=tmp(vars);
   alias(tmp=tmp(vars))
else
   tmp1:=op(0,tmp);
   convert(tmp1,string);
   print(cat(",` has already been declared`));
   print(cat(`Do you wish to redefine `,
             ",`? y or n`));
   if readline() = 'y' then
      if not {vars} union {op(tmp)} = {op(tmp)} then
         print(`WARNING: The new variables`);
         print(vars);
         print(`are not a subset of the old variables`);
         print(op(tmp))
      fi;
      member(tmp1=tmp,[_FUNCNAMES],'pos');
      _FUNCNAMES:=op(subsop(pos=NULL,[_FUNCNAMES]));
      alias(tmp1=tmp1);
      assign(tmp=tmp1(op(vlist)));
      _FUNCNAMES:=_FUNCNAMES,tmp1=tmp1(op(vlist));
      alias(tmp1=tmp1(op(vlist)))
   fi
fi
od end:
```

References

[1] A. C. Hearn. REDUCE *user's manual. Version 3.4*, Rand Publication CP78, Rand Corporation, Santa Monica, 1991.

[2] M. S. Hickman. *The Use of* MAPLE *in the Search for Symmetries*, Mathematics Department Research Report No. 77, University of Canterbury, New Zealand, 1993.